

APPENDIX A  
PSEUDO-CODE EXAMPLE

```

1
2
3 /
4 / This is a sample pseudo-code program that demonstrates a delayed
5 / acknowledgement mechanism for implementing fault tolerance.
6 / Note this is just one representative implementation of the methodology.
7 / It assumes the availability of certain middleware capabilities through
8 / several hypothetical middleware library calls.
9 /
10 / Authors: Mingqiu Sun and Mahesh Bhat
11 / Date: 5/10/2001
12 /
13 /*****
14
15 #include <comutil.h>
16 #include <iostream.h>
17
18 /*****
19 / The following class represents callback functions that handle client requests.
20 /
21 /*****
22 class EventHandler
23 {
24     App* m_pMApp;
25     TargetApplication *targetApp; // to perform a task in a workflow sequence
26
27 public:
28     EventHandler(App* pApp)
29     { m_pMApp = pApp; }
30
31     virtual ~EventHandler()
32     {}
33
34 private:
35     /*****
36     / The following callback function implements a simple workflow sequence that has
37     / several tasks: 1. connect to an application; 2. pass the message from client
38     / to the application; 3. finish processing and send confirmation back to client.
39     /*****
40     virtual void onEvent( const Event& refEvent)
41     {
42         const Event* pEvent = refEvent.getEvent();
43         if (pEvent) {
44             String sMessage = pEvent->getMessage();
45
46             //This corresponds to Task T1 (FIG. 4) where it is trying
47             //to connect to the application.

```

```

49         int connected = targetApp.connect();
50
51         //This call corresponds to T2 (FIG. 4).
52         //Here it tries to load the message in the application.
53
54         int loaded = targetApp.loadData (sMessage);
55
56         //This corresponds to final task TFT (FIG. 4) where it checks if
57         everything //is fine.
58         if(loaded)
59         {
60             //Trigger the confirmation
61             int triggered = pEvent.trigger();
62
63             //if triggered, then send conformation back to the client
64             //this action could be performed by a different process
65             if (triggered)
66                 pDataEvent->sendConfirmationBack();
67         }
68         // else client would resend request back due to a lack of confirmation
69
70     }
71 }
72 }; // EventHandler
73
74
75
76 /*****
77 / The following class represents a process that handles incoming client requests. It
78 / launches a certified messaging listener on start, which in turn launches a callback
79 / function to start a workflow sequence upon arrival of a client request.
80 /
81 /*****/
82 class MessageProcessor : public App
83 {
84 public:
85     MessageProcessor( AppProperties* pAppProperties);
86     ~MessageProcessor();
87
88 protected:
89     virtual void onStart() throw(Exception);
90     virtual void onEnd() throw(Exception);
91
92 private:
93     Subscriber*           m_pMSubscriber;
94     DataEventHandler*      m_pDataEventHandler;
95 }; // MessageProcessor
96
97
98 MessageProcessor::MessageProcessor( AppProperties* pAppProperties)
99 :App(AppProperties)

```

```

100  {
101  }
102
103  MessageProcessor::~MessageProcessor()
104  {
105  }
106
107  void
108  MessageProcessor::onStart()
109  {
110      MProperties* pMProps = App::getProperties();
111
112      // subscriber
113      MsgSubscriber* pMSubscriber = getMsgSubscriber();
114
115      m_pEventHandler = new EventHandler( this);
116      pMSubscriber->addProcessor( m_pEventHandler );
117  }
118
119  void
120  MessageProcessor::onEnd()
121  {
122      delete m_pEventHandler;
123      delete m_pMSubscriber;
124  }
125
126
127
128
129  /*****
130  / The following is the main program that creates a MessageProcessor
131  /
132  /*****
133  int main(int argc, char** argv)
134  {
135      String appName = "testMessageProcessor";
136      String appVers = "3.0";
137
138      try
139      {
140          AppProperties appProps;
141          appProps.setAppName(appName);
142          appProps.setAppVersion(appVers);
143
144          MessageProcessor* pMessageProcessor = new MessageProcessor(&appProps);
145
146
147          //This corresponds to the TST process in the workflow sequence.
148          pMessageProcessor->start();
149
150          // exiting from main loop - perform cleanup and exit

```

```

151         delete pMessageProcessor;
152     }
153     catch(Exception& e )
154     {
155         String s = e.getType();
156         cerr << s << endl;
157     }
158     return 0;
159 } // main
160
161

```